

### Amendments to the Specification

**Amend the paragraph found on page 19 from lines 9-16 as follows:**

This optimization allows processes to vote for the value of the coin before they know it by replacing the vote for Y or N by a vote for "coin". The receiver is then able to evaluate the vote as soon as the corresponding coin is opened. This allows to piggyback the coin onto the pre-vote, saving one message per round. If threshold signatures are used, the way of signing messages has to be changed. Instead of signing a message "Party A votes Y" two messages "Party A votes Y if the coin is N" and "Party A votes Y if the coin is Y" are signed. Correspondingly, if a party wants to vote "coin", it ~~singes~~ signs two messages "Party A votes N if the coin is N" and "Party A votes Y if the coin is Y". This way, it is guaranteed that the threshold signatures can still be applied.

CH919990046-US1

-2-

**Amend the paragraph found on page 20, lines 2-15 as follows:**

There are parties which do not terminate the protocol. It is possible that some parties decide in round  $R$ , going through the voting part one more time to help the other parties decide. The other parties that decide in round  $R + 1$ , however, also try to vote one more round. But waiting for these votes may never terminate as more than  $t$  parties may already have quit the protocol. Though it is not a major problem, all the afflicted parties have to keep some process running in the background which keeps on waiting for more votes that will never arrive. This is still inconvenient. One solution to this is that if one party decides, it does not participate in further rounds of the protocol. Instead, it sends a message to all other parties containing its agree-value  $\in \{Y, N\}$  and  $n-t$  signatures on main-votes that led to this decision. All parties that receive this message immediately stop the protocol and echo this message to all other parties. This allows the one party to completely terminate the protocol after broadcasting this decision message. As this model assumes eventual message delivery, which is not present in a real-world scenario where messages can be lost, some lower-lever protocol would still have to be able to resend ~~resent~~ the message, but this is unavoidable and requires far less effort than keeping the entire protocol running.

**Amend the paragraph found on page 21, lines 16-26 as follows:**

FIG. 4a shows a schematic illustration of a fully connected asynchronous network without routers. Each circle indicates a party or a participating network device, whereby each party communicates with all the other parties. A distinct subset  $R$  of  $t + 1$  processes is defined and called "relays". FIG. 4b shows a schematic illustration of a connected network having two relays or relay-stations 40. If a first-party 41 wants to broadcast a message  $m$  to all parties, the first-party 41 sends  $m$  only to the relay-stations 40. All these relay-stations 40 then wait until they collected  $n-t$  valid messages for this transaction-identifier  $TID$  and round  $r$ . They then evaluate the messages and send the verifiable result to all parties that are not  $[[a]]$  relay-stations 40. The number of messages needed for  $n$  broadcasts is  $(n-1)(t+1)$  for sending the messages to the relays ( $n-1$  because a relay does not send a message to itself) and  $(t+1)(n-(t+1))$  for the relays to send the messages to all other parties.